thinkScript

User Manual

Need more help? Email us at support@thinkorswim.com

You may also find it helpful to join this online discussion group started by existing thinkScript users. thinkScript discussion group

---

thinkScript Introduction

Basic Commands

- DECLARE
- PLOT
- DEF
- INPUT
- REC

Using Functions

Function Definitions

thinkScript Advanced & Saving

NEW! - Limiting input data (ENUM & SWITCH)

Accessing Data From Another Symbol

Using Historical Data

Referencing Pre-Defined Studies

Choosing Colors

Saving Your Study

thinkScript and Strategies

Adding Strategies

Editing Strategies

- A Simple Example
- Using Functions
- Creating Input Parameters
- Referencing Historical Bars
- Accessing External Price Data
- Choosing Colors

Saving Your Strategy

Introduction

thinkScript is a basic editing tool used for fine-tuning an existing study or implementing your own study on the Charts tab. Put simply, thinkScript is a way to manipulate the closing, opening, high and low price of a stock or index, as well as the trading volume and volatility of a stock or index with a formula and then display the results on a chart. One or all of those data types (open, high, low, close, volume and imp_volatility) in conjunction with some basic functions will constitute the building blocks of your thinkScript studies. Like building blocks, each element of thinkScript is, in itself, fairly simple but combining them together can yield some complex and powerful results.

To begin using the thinkScript editor for studies, go to the Charts tab, click 'Studies' in the upper right corner of the page and select 'Edit Studies'. This will open the 'Edit Studies' window. To use thinkScript to make a study of your own you now have two choices:

- To create a new study by fine-tuning an existing study, first find a study in the 'Available studies' list marked by the icon. These studies are available for copying. Next click on the Blue Bullet next to the chosen study and click 'Copy...'. This will open the 'New study' editor with the chosen study's definition preloaded.
- To create your own study from scratch, simply click the 'New study' button in the bottom left corner of the 'Edit Studies' window. Doing so will open a blank 'New study' editor.

PLEASE NOTE: Every line of thinkScript code must end with a semi-colon(;). Also, thinkScript code is not case-sensitive.

Basic Commands

thinkScript supports five simple commands: DECLARE, PLOT, DEF, INPUT and REC. These commands control the basic behavior of your thinkScript study.

Basic Command - DECLARE

The DECLARE keyword is a method for telling the chart something basic about the appearance of the study you are creating. There are three current uses for declare:

```
declare upper;
```

This command places your study in the main chart window on top of the pricing bars. You want an upper study if your study plot falls more or less within the same range as pricing. 'SimpleMovingAverage', for instance. By default, all studies created with thinkScript are upper studies and you don't have to make this declaration.

```
declare lower;
```

This statement places your study on a separate graph under the pricing bars. You want a lower study if your study is some complex value in the range of 0-100, i.e. not directly related to the market values.

```
declare hide_on_intraday;
```

This command obscures your study for pre- or post-market (non-regular market hours) data on intraday graphs. This is useful if you use IMP_VOLATILITY where there is no implied volatility

outside of regular market hours because options do not trade then even though a stock may.

Basic Command - PLOT

The PLOT command actually renders the data you are working with on the chart. Every study contains at least on plot declaration (without one, nothing would be displayed on the chart). For it to work, you must give the data you are trying to plot a name and then define what that data will represent. For example, if you want to plot the closing price of a stock, you can't simply write

```
plot close;
```

That won't work. You have to provide both the name and the data:

```
plot price = close;
```

The name for what you are plotting can be anything that makes sense to you, such as

```
plot median = (open + close) / 2;
```

This script will display a line that adds the opening and closing price of each bar of data and divides the sum by two.

You can make multiple plot declarations to show more than one line on the chart, such as

```
plot median = (open + close) / 2;
plot ohlc = (open + high + low + close) / 4;
```

As you can see, the plot command uses the basic information of open, high, low, close, volume and implied volatility in mathematical formulas and displays the result on the chart.

Basic Command - DEF

The DEF command defines a variable you'd like to work with. For example, you can define something called "MyClosingPrice" as

```
def MyClosingPrice = Close;
```

By using the def command, you can create a variable that can be used in another formula in the study. This let's you construct complex formulas from simpler elements. For example, if you wanted to divide the range of a stock price by the closing price, you could create a variable for the range:

```
def range = high - low;
```

then create another variable

```
def percentrange = range / close;
```

Doing it this way lets you re-use variables and gives you the power to combine complex variables in your formulas more easily. By defining a variable such as

```
def SMA = average(close, 20);
```

you can use that variable "sma" in your code as part of another calculation. For instance,

```
plot doubleSMA = SMA * 2;
plot tripleSMA = SMA * 3;
```

This will display lines that are 2 and 3 times the SMA variable. In this case, SMA itself is not displayed on the chart because there is no plot declaration to do so.

Basic Command - INPUT

Most studies are adjustable in terms of length, bounds or levels. You can create an adjustable parameter for your thinkScript study using the INPUT command. An input is like a def that can be adjusted in the 'Edit Studies' window or in the 'Format Study' dialog found by right-clicking on a study line right on the graph. For instance,

```
input length = 12;

input price = close;


plot SMA = average(data = price, length = length);
```

Here '12' and 'close' are default values which you can override on the preferences panel the way you adjust any pre-defined studies.

Basic Command - REC

The REC command lets you reference a historical value of a variable that you are calculating in the study itself. Rec is short for "recursion". For example,

```
def mystudy = mystudy[1] + 5;
```

won't work because you are trying to use a previous value of the variable "mystudy" in the formula.

```
rec mystudy = mystudy[1] + 5;
```

will work because the rec command allows you to use previous values of the variable in the formula.

Using Functions

thinkScript has 12 mathematical and logic functions (AbsValue, average, between, ceil, if, isNaN, log, max, min, sqrt, sum and totalsum) that can be performed on the price data to generate the desired study. For example,

```
plot CloseAvg = average(close, 12);
```

displays an average of the last 12 days' closing prices.

Each function has required parameters. 'Average', for example, requires data (open, close, etc.) and length (number of bars). You can specify these parameters in any order. For example,

```
plot SMA = average(data = close, length = 50);
```

and

```
plot SMA = average(length = 50, data = close);
```

will show the same result on the chart.

Function Definitions

AbsValue calculates the absolute value of some value. For example,

```
plot Absolute = AbsValue(low - high);
```

will plot the absolute value of the difference between the low and high price (which, by definition, is a negative number).

average calculates the average of a set of data. For example,

```
plot SMA = average(close, 20);
```

will display a moving average of the last 20 closing prices. If you do not specify a length for the data set, the default number of bars is 12.

between is a logic function that gives a value of 1 (true) if the data is between two parameter values, and 0 (false) if the data is outside of the two parameter values. The first parameter is the data type to compare on. For example,

```
plot center = between(close, 140, 160);
```

will display a 1 if the closing price is between 140 and 160, and 0 if the closing price is below 140 or above 160. "Between" is mainly used as part of a test within a larger study.

ceil rounds a value up to the nearest integer. For example

```
plot data = ceil(close);
```

displays a line that finds the integer (whole number, no fractions) that is equal to or next higher than the close price.

if can be used in two ways. First, it can be used on the right side of an equation bu using 3 parameters: a condition, a true value and a false value. For example,

```
plot resistance = if(close > 150, 150, 100);
```

will draw a line that is at a value of 150 if the closing price is 150 or higher, and a value of 100 if the closing price is less than 150.

Secondly, 'if' can be used in conjunction with 'else' to create more complex conditions:

```
rec upper;
```

```
if open == close then {
   upper = upper[1] + 2;
} else {
   upper = upper[1] + 1;
}
```

Obviously, this example could be extended to perform even more calculations for each condition simply by adding additional lines inside each. Please note the placement of the '{}' brackets in this example. Also note that when evaluating equality in an 'if' statement, two equal signs must be used ('==').

Like between, the 'if' function will most likely be used as a test for a larger study.

isNan stands for "is Not a Number" and generates a 'true' value if you try to take the square root of a negative number or divide by 0. For example,

```
data.assignValueColor(if isnan(sqrt(-20)) then color.white else color.red);

data.assignValueColor(if isnan(0 / 0) then color.white else color.red);
```

log is a mathematical function that calculates the natural log of a number.

```
plot data = log(close);
```

will draw a graph of the log of the closing price of a stock.

max finds the greater of two values. For example,

```
def SMA = reference simplemovingavg;
plot data = max(close, SMA);
```

displays the higher of either the closing price or the simple moving average.

min finds the smaller of two values, and is the opposite of "max". For example,

```
def SMA = reference simplemovingavg;
plot data = min(close, SMA);
```

displays the lower of either the closing price or the simple moving average.

sqrt is a mathematical function that calculates the square root of a number.

```
plot data = sqrt(close);
```

will draw a graph of the square root of the closing price of a stock.

sum adds up the values of a set of data. For example,

```
plot data = sum(close, 20);
```

will display a line that is the sum of the last 20 days' closing prices.

```
plot data = sum(close, 20)/20;
```

will display a line that is the sum of the last 20 days' closing prices divided by 20. This actually gives you a 20 day moving average. If you do not specify a length for the data set, the default 12 bars of data is used.

totalsum adds up all the values that are available on the chart, and lets you see a running

total of something. For example,

```
plot data = totalsum(volume);
```

will show a line that is the total accumulated volume for the time frame that is on the chart. If the chart displays one year of data, totalsum will add up year. If you are looking at one month of data, totalsum will add up one month.

Limiting Input Data (ENUM & SWITCH)

Let's say you wanted to create an input editable from the 'Edit Studies' window but, for ease of use, you wanted to limit the possible answers for that input to a predetermined set (for instance, the trading days of the week). In an application or software such an input is called a 'select box'. You can create a select box in thinkScript by using a construction known as an 'enum'. This is simply a predefined set of data and the code looks like this:

```
input day = {default Mon,Tue,Wed,Thu,Fri};
```

Please notice that one of the elements in the set must be defined as the default value. In this case it's 'Mon'. The value generated for 'Mon' by the select box is actually 0 and each succeeding value is 1 greater ('Tue' = 1, 'Wed' = 2 ... 'Fri' = 4).

In order to do something more powerful with the results of your select box rather than simply operate on the numeric value (0,1,2...), you can use a SWITCH statement. A SWITCH allows you to define a discreet action for each 'case'. For example,

```
input day = {default Mon,Tue,Wed,Thu,Fri};
plot vertical_line;
```

```
switch(day) {
 case Mon:
  vertical_line = average(close, length = 8);
 case Tue:
  vertical_line = average(close, length = 13);
 default:
  vertical_line = average(high);
}
```

In this case, specific plots will be drawn if you input 'Mon' or 'Tue' and a third drawn if you input 'Wed', 'Thu' or 'Fri' as defined by the default case. At this point, you should begin to see how you could use ENUM and SWITCH to use predefined data sets like the months of the year, etc.

Accessing Data From Another Symbol

To access data from another symbol in your code, append the name of the symbol in quotes and parentheses to the data type you want to use. For instance,

```
plot data = close - close("GOOG");
```

will give you a line that is the difference between the closing price for the symbol that you have in the chart and the closing price for Google (GOOG).

Using Historical Data

To access a value from a previous bar of data, you can use what is called [offset] syntax. For

example, close[1] will give you the closing price from one day ago; close[2] will give you the close 2 days ago, etc.

```
plot data = close - close[5];
```

will display a line that is the difference between the current closing price and the closing price 5 days ago. Please note: in thinkScript, a positive number is used to refer to data in the past. Negative numbers will give you bars in the future when working from historical data.

Referencing Pre-Defined Studies

thinkScript allows you to reference studies that are already available on the Charts tab in your code. You use the reference keyword to do this.

```
plot SMA = reference simplemovingavg;
```

or

```
plot SMA = simplemovingavg();
```

will plot a simple moving average with the default parameters of that study. So long as the study name is followed by parentheses, you don't need to include the word 'reference'. You can change the parameters of the study within reference by inputting them between parentheses.

```
plot SMA = simplemovingavg(price = volume, length = 20);
```

will plot a 20 day moving average of the volume. To see the input parameters of a particular study, click on that study in the 'Edit Studies' section of the Charts tab. You will see the available parameters listed in the study properties section.

By default, the first plot of a study is always referenced. However, you can specify the plot you want to reference, as well.

```
plot bear = reference ElderRay."BearPower"

plot bear = ElderRay(length = 6)."BearPower"
```

Choosing Colors

Each plot has its main color which you can change using the setDefaultColor function:

```
plot diff = close - close[1];
diff.setDefaultColor(getcolor(5));
```

In this example, the 5th color of a dynamic 'look & feel' palette was used. These colors will always be part of the Color Scheme you chose to run thinkDesktop with - Black, White or Metal. You can also choose explicit colors:

```
plot diff = close - close[1];
diff.setDefaultColor(Color.Red);
```

Below the 'Functions' section in the right sidebar you will see that the definitions of some Constants are also provided. There you will find all of the standard colors you can use in conjunction with the 'Look & Feel' functions.

Lastly, you can override plot main color using assignValueColor function:

```
plot diff = close - close[1];
diff.assignValueColor(if diff >=0 then Color.UPTICK else Color.DOWNTICK);
```

Saving Your Study

When you are done with your code, you can save it and apply it to your chart. At the top of the "New Study" editor is a field where you can name your study. Once you've provided the name, click the "Save" button and your new study can be applied to any chart like any of the pre-programmed studies.

If you want to change the code in your custom study, return to the "Edit Studies" window by clicking the "Studies" button in the upper right hand corner of the Charts tab, then clicking on "Edit Studies". Locate your study in the "Available Studies" list. Custom studies are marked with the icon. Click the blue bullet to the left of the name of the study and select "Edit source". That will return you to the study editor to make any changes.

---

thinkScript and Strategies

Just like with studies, thinkScript can be used for fine-tuning pre-defined strategies or creating strategies of your own in the TOS Charts charting package. When strategies are drawn on a chart, buy and sell triggers appear in response to the conditions defined in the strategy. To begin using the thinkScript editor for strategies, go to the TOS Charts tab, click 'Studies' in the upper right corner of the page and select 'Edit Strategies'. This will open the 'Edit Strategies' window.

Adding Strategies

On the 'Edit Strategies' window you'll see a list of predefined strategies under 'Available

Strategies'. These strategies are actually based on studies and render buy and sell triggers when their underlying study meets a certain condition. If you simply want to draw predefined strategies on the chart, right click the Blue Bullet and select 'Add Strategy'. This will cause the strategy to appear in the 'Added strategies' box and any of its customizable inputs to appear under 'Strategy properties'. **PLEASE NOTE:** in order to render strategies on your chart, you must select BOTH the buy and sell ends. These are indicated by the letters SE (Short End) and LE (Long End) in the strategy name. Once you've added your strategies and adjusted their properties click 'OK' (or 'Apply' to keep working in the window) to draw them on your chart.

Editing Stategies

More than likely, you will want to create strategies of your own rather than use predefined strategies. To do so, you have two choices:

- To create a new strategy by fine-tuning an existing strategy, first find a strategy in the 'Available strategies' list. Next click on the Blue Bullet next to the chosen strategy and click 'Copy...'. This will open the 'New strategy' editor with the chosen strategy's definition preloaded.
- To create your own strategy from scratch, simply click the 'New strategy' button in the bottom left corner of the 'Edit Strategies' window. Doing so will open a blank 'New strategy' editor.

It is important to remember that most strategies are based on movement in an underlying study. If you haven't done so, it might be good to familiarize yourself with how thinkScript

handles studies above.

Editing Stategies - A Simple Example

Here are examples of the buy and sell sides of a very simple strategy. In this example, the user simply wants to be able to set a buy (entry) trigger when the closing price drops below 50 and a sell (exit) trigger when the price exceeds 70:

```
BUY SIDE -

Name: myEntry

Body:

 declare ENTRY;

 addOrder(close < 50);

SELL SIDE -

Name: myExit

Body:

 declare EXIT;

 addOrder(close > 70);
```

For now, let's just take a look at the body of these definitions - the name has more to do with how you save the strategy. As you can see, the script involved here is very simple. First, the strategy needs to declare if it is BUY (entry) or SELL (exit) side. Then, it uses a simple function, *addOrder*, which indicates that we want to set a trigger when the condition inside the parentheses is met. In this case, that condition is either 'close < 50' or 'close >70'.

Editing Stategies - Using Functions

In the previous example, we used a simple function, *addOrder*. *AddOrder* takes one required parameter, 'condition' (e.g. 'close < 50'), and one optional parameter, 'price'. Price can be set using a variable like 'open' or a value like '52'. Definitions for all of the available functions can be found by holding your mouse over the function name in the edit page or in the sidebar on the right side of the 'New strategy' editor. There you'll find functions grouped by category - 'Technical analysis', 'Mathematical' and 'Look & Feel'. Click on a function name to see details of how that function works in the lower box on the right-hand sidebar. Clicking on 'Mathematical' > 'average', for instance, reveals that this function returns the average value of a dataset. No surprise there.

In every respect using functions in strategies behaves the same as using them when editing studies. See 'Using Functions' for details.

Editing Stategies - Creating Input Parameters

Most of the time you'll want to create strategies that allow you to easily adjust the boundaries of the strategy. You can do this by creating an *input* and its default value in the thinkScript definition. These inputs can be adjusted in the 'Edit Strategies' window (HINT: You can also open the 'Edit Strategies' window by left-clicking on the strategy from a chart). If you want to create an adjustable parameter for your thinkScript strategy, you use the *input* keyword:

```
input parameter_name = default_value;
```

for instance,

```
input length = 12;

input price = close;


def delta = price - price[length]
```

Here 12 and close are default values which you can override on the preferences panel the way you adjust pre-defined strategies.

Editing Stategies - Referencing Historical Bars

In the example above you see a construct that looks like this:

```
price[length]
```

Given our default values for price and length, this would actually resolve to 'close[12]'. What you see here is a reference to a bar other than current bar using *[offset]* syntax. In this case, 'close[12]' would give you the close price on the bar 12 bars in the past. It's important to remember that positive values are bars BACK and negative numbers are bars FORWARD:

```
def diff = close - close[-1];
```

In this example, diff equals the difference in the current and the *next* value.

Editing Stategies - Accessing External Price Data

For comparison strategies, you can access 'external' symbol market data like this:

```
def diffGoog = close - close("GOOG");
```

Editing Stategies - Choosing Colors

Each strategy can be assigned a color using the *setColor* function:

```
setColor(getColor(4));
```

In this example, the 4th color of a dynamic 'look & feel' palette was used. These colors will always be part of the Color Scheme you chose to run thinkDesktop with, Black, White or Metal. You can also choose explicit colors:

```
setColor(Color.Red);
```

Below the 'Functions' section in the right sidebar you will see that the definitions of some Constants are also provided. There you will find all of the standard colors you can use in conjunction with the 'Look & Feel' functions.

Saving Your Strategy

Once you are done editing the strategy, you can save it so that it will be available to add to the charting package from the 'Edit Strategies' window. At the top of the 'New Strategy' editor is a field where you can name the strategy. You can name your new strategy the way you like it or leave the pre-defined 'NewStrategyXX' name. Once you've provided the name, click 'Save' and your new strategy will be available for drawing over any chart. To do so, add it to the list of 'Added strategies' from the 'Edit Strategies' window and click 'OK' (or 'Apply' to keep working in the window).

If you need to make changes to your custom strategy, return to the 'Edit Strategies' window by clicking 'Studies' in the upper right corner of the TOS Charts tab. Locate your strategy in the 'Available strategies' list. Custom strategies are marked with the icon. Double-click the strategy or click the Blue Bullet to the left of the strategy and select 'Edit source...'. This will

return you to the strategy editor to make any necessary changes.